Parallel Depth–First Search in Directed Graphs

Alok Aggarwal
IBM Research Division
T. J. Watson Center, Box 218
Yorktown Heights, NY 10598

Richard J. Anderson
Dept. of Computer Science
University of Washington
Seattle, WA 98195

and

Ming Y. Kao
Dept. of Computer Science
Indiana University
Bloomington, IN 47405

TECHNICAL REPORT N0. 250

Parallel Depth—First Search in Directed Graphs

By

Alok Aggarwal, Richard J. Anderson & Ming Y. Kao

May, 1988

Parallel Depth–First Search In Directed Graphs

Alok Aggarwal
IBM Research Division
T.J. Watson Center, Box 218
Yorktown Heights, NY 10598

Richard J. Anderson
Dept. of Computer Science
University of Washington
Seattle, WA 98195

Ming Y. Kao
Dept. of Computer Science
Indiana University
Bloomington, IN 47405

(Extended Abstract)

Abstract

In this paper we show that a directed depth first search tree can be constructed by a random NC algorithm. Previous results on parallel depth first search had only applied to undirected graphs. The algorithm relies on finding directed cycle separators. A cycle separator of an n vertex directed graph, is a directed simple cycle such that if the vertices of the cycle are deleted from the graph then the size of the largest strongly connected component in the resulting digraph is at most $\lfloor n/2 \rfloor$.

1 Introduction

The directed depth-first search problem is: given a digraph $G = (V, A)$ and a vertex x, construct a directed forest F that corresponds to conducting a depth-first search of the graph starting from the vertex x. In this paper we present a fast parallel algorithm for constructing a depth-first search tree of a directed graph. The problem of performing depth-first search in parallel has been considered by a number of authors [RC78, EA77, Re83, GB84, An85, AA87, Sm83, Ka88, HY87]. Reif [Re83] showed that computing the lexicographic depth-first search forest is P-complete; Ghosh and Bhattacharjee [GB84] provided an NC algorithm for directed acyclic graph; Smith [Sm83] provided an NC algorithm for undirected planar graphs; Aggarwal and Anderson [AA87] gave an RNC algorithm for general undirected graphs; and Kao [Ka88] recently provided an NC algorithm for directed planar graphs. Although, Aggarwal and Anderson [AA87] provided an RNC algorithm for general undirected graphs, they could not settle whether a depth-first search forest of a directed graph could be found in RNC. In this paper, we settle the question by providing an RNC algorithm for this problem.

This algorithm uses a similar divide and conquer strategy as was used for undirected graphs [AA87]. At the very highest level, the algorithm is to find a portion of the depth first search tree and then perform recursive searches of independent components of the graph. The key idea in dividing the problem is to use a *directed cycle separator*. Although this algorithm uses a similar

approach to that of the undirected algorithm, there are some substantial differences. First of all, directed graphs introduce many subtleties. The distinction between strong and weak components plays a major role in a portion of the algorithm. The graph theory is quite a bit more involved than in the previous algorithm. The second major difference arises in the construction of the separator. Both the directed and undirected algorithms construct separators by repeatedly joining paths until only a small number of paths remain. However, a key idea in the undirected case is to traverse a path in one of two possible directions to minimize the size of segments that cannot be joined. In the directed case, it is not possible to choose the direction of traversal, so a different approach is necessary. The third major difference is in the application of the separator that allows the problem decomposition. In the undirected case, the decomposition of the graph was almost immediate, given the appropriate separator. However, with directed graphs, there is a fair amount of work that is necessary to successfully reduce the size of the graph for the recursive calls.

In the next section we introduce the directed cycle separator and discuss a number of preliminary concepts. Section 3 gives the first major result of the paper, establishing an NC-equivalence between finding separating cycles and solving depth first search in directed graphs. Section 4 shows how to reduce the problem of finding a cycle separator for a given directed graph to a collection of matching problems. The matching problems can be solved in RNC. Section 5 puts the results together, discusses run time and processor bounds, and discusses open problems and extensions.

2 Cycle Separators

Most of the work on parallel depth first search and related problems has relied on some form of graph separator. A graph separator is a subgraph whose removal disconnects the graph in some manner. The maximal path algorithm [An85] and undirected depth first search algorithm [AA87] used path separators. The algorithms for depth first search of planar graphs have used various forms of cycle separators [Sm83, HY87, Ka88]. Our definition of a *separator* of an n vertex directed graph $G = (V, E)$ is a set of vertices V', such that $G(V - V')$ has no strongly connected component containing more than $\lfloor n/2 \rfloor$ vertices. A *directed path separator* is a set of vertices V' that forms a directed simple path in G. Similarly, a *directed cycle separator* is a set of vertices V' that form a simple directed cycle in G. We consider a single vertex to be a cycle of length zero. Thus, if the removal of a vertex separates the graph, we will considerate it a cycle separator.

The natural sequential algorithms for constructing path separators relies on depth first search. The algorithm can be modified to construct a separating cycle, so we have:

Observation 2.1 *Every directed graph has a cycle separator. Furthermore, this separator can be found in linear sequential time.*

Proof: Omitted. ∎

In our algorithm, we will use directed paths (dipaths) and vertices extensively. We often use a set of vertices to denote an induced digraph wherein the arcs are inherited from the given digraph $G = (V, A)$. A dipath is an ordered set of vertices $p = p_1, ..., p_k$ with arcs $(p_i, p_{i+1}) \in A$ for $1 \leq i < k$ (where p_i points to p_{i+1} in the arc (p_i, p_{i+1})). A *lower segment* of p is the dipath $p_1, ..., p_{\lfloor k/2 \rfloor}$ and the *upper segment* of p is the dipath $p_{\lfloor k/2 \rfloor + 1}, ..., p_k$. For a set of dipaths $Q = \{q_1, ..., q_m\}$, we use $m = m$ to denote the number of dipaths in Q. In unambiguous cases, we will often refer to a dipath when, in fact, we imply the vertices of that dipath. For example, if p is a dipath and Q a set of

dipaths, then $G(V-p)$ will denote the induced subgraph where all the vertices of p have been removed, and $G(V-Q)$ will denote the induced subgraph where all the vertices contained in the paths of Q have been removed. The observation given below demonstrates the existence of a cycle separator in any digraph; a similar observation also holds for undirected graphs but is omitted for the sake of brevity.

3 Using a Cycle Separator to Find a DFS Tree

From the point of view of parallel computation, observe that given a depth-first search tree of $G=(V,A)$, we can use the proof of Observation 1 to obtain a directed cycle separator (or a dipath separator) in $O(\log^2 n)$ parallel time by using n^3 processors on a PRAM. Consequently, in this section, we show that if a cycle separator can be found in NC then so can a depth-first search tree. More precisely, we give a procedure that finds a depth-first search tree of $G=(V,A)$ in $O(\log^2 n(T_c(n)+\log^2 n))$ time using $[P_c(n)+n^3]$ processors where $T_c(n)$ and $P_c(n)$ denote the time and the processor complexity of finding a cycle separator of G. In the next section, we will show how to find a cycle separator efficiently on a PRAM.

Given a cycle separator of G and a vertex $r \in G$, it can be readily seen that a dipath separator, P_r, that begins at at r, can be found in $O(\log^2 n)$ time by using n^3 processors. (These processor and time bounds can be achieved, for example, by computing the transitive closure of a suitable digraph.) However, if we remove this path separator from G, then although the largest strongly connected component has at most $\lfloor n/2 \rfloor$ vertices, the largest weakly connected component may have almost n vertices. Consequently, by removing P_r we have reduced the size of the strongly connected components in G by a half but the sizes of the weakly connected components in $(G-P_r)$ may still be quite large. In view of this, we discuss below a subroutine that removes a set of dipaths from G (which, in fact, will form the paths in the final depth-first search tree) such that the remaining digraph has no weakly connected component of size greater than $\lfloor n/2 \rfloor$.

We call a digraph, G, *rooted* at a vertex if this vertex can reach all other vertices of G using directed paths. A *partial* DFS tree in a rooted digraph is a subtree of a DFS tree such that the digraph and both on trees are rooted at the same vertex. Let T be any partial DFS tree of G; let $x_1, x_2, ..., x_t$ be the vertices of T listed in the DFS *last visit* order (i.e., if this partial DFS tree is traversed sequentially then x_{i-1} and its descendants would be visited before we *backtrack* to x_i). Furthermore, let $y_{i,1}, ..., y_{i,k_i}$ be all the end vertices not in T of the arcs whose starting vertex is x_i, but not on x_l for $1 \leq l \leq i-1$; the order of $y_{i,1}, ..., y_{i,k_i}$ being arbitrary. For any two integer pairs, (i,j) and (i',j'), we say $(i',j') < (i,j)$ if either $i' < i$ or $i = i'$ and $j' < j$. For any digraph D and a vertex $x \in D$, let $R(x,D)$ denote the set of vertices that can be reached from x using the directed paths in D. We call a subgraph of G, a *dangling subgraph* $DSG((i,j),T)$ with respect to (i,j) and T if it is formed by vertices in $G-T$ that are reachable from $y_{i,j}$ but not from $y_{i',j'}$ for any $(i',j') < (i,j)$. That is, $DSG((i,j),T) = R(y_{i,j}, G-T) - \bigcup_{(i',j')<(i,j)} R(y_{i',j'}, G-T)$. Observe that a depth-first search tree of G is simply the union the arcs in a partial depth-first search tree, T, of G, the set of arcs $(x_i, y_{i,j})$ for which $DSG((i,j),T)$ is non-empty, and an arbitrary depth-first search tree rooted at $y_{i,j}$ for each non-empty $DSG((i,j),T)$.

Consider a partial depth-first search tree, T, such that $x_1, ..., x_t$ are its vertices in the last visit order and every strongly connected component of any dangling subgraph $G-T$ has at most $\lfloor n/2 \rfloor$ vertices. Now, for a dangling subgraph in $G-T$, it is possible that although the strongly connected components are appropriately small, there may be a weakly connected component with almost

n vertices. Consequently, we call a dangling subgraph *heavy* if it has more than $\lfloor n/2 \rfloor$ vertices, otherwise, we call it *light*. Now, if there is no heavy dangling subgraph then we can compute a depth-first search tree by recursively computing the depth-first search trees in the non-empty dangling sub-graphs of $G - T$ that are rooted at $y_{i,j}$ for $1 \leq i \leq k$ and $1 \leq j \leq k_i$, and by adding the remaining appropriate edges. Consequently, assume that there is at least one heavy dangling subgraph. Because all such subgraphs are vertex disjoint, there is exactly one heavy subgraph, call this $DSG((i_o, j_o), T)$. Let H denote the weighted subgraph induced by *contracting* the strongly connected components of $DSG(i_o, j_o)$ where a vertex in H has weight w if there are w vertices in the corresponding strongly connected component of $DSG((i_o, j_o), T)$. Since $DSG((i_o, j_o), T)$ is heavy, there exists a vertex in H such that the weight of this vertex and its descendants is greater than $\lfloor n/2 \rfloor$ each of its child and its descendant has weight at most $\lfloor n/2 \rfloor$. Call such a vertex, $s_w \in H$, a *splitting vertex*; call the corresponding strongly connected component in $DSG((i_o, j_o), T)$ a, *splitting component*, and denote this splitting component by G_w. Now, using the subroutine for finding a cycle separator, find a cycle separator in G_w and let this be C_w. Also, find a path P_o in $DSG((i_o, j_o), T)$ that goes from y_{i_o,j_o} to any vertex of C_w and let the new tree be $T_1 = T \cup (x_{i_o}, y_{i_o,j_o}) \cup P_o \cup C_w$. Observe that T_1 is a partial depth-first search tree of G with the largest strongly connected component of $G - T_1$ having at most $\lfloor n/2 \rfloor$ vertices. Also, if there is a heavy dangling subgraph in $G - T_1$ then the splitting component in this subgraph is, indeed, a subgraph of G_w and this subgraph has at most $\lfloor w/2 \rfloor$ vertices. Consequently, after $\lceil \log w \rceil$ such phases of updating the partial depth-first search tree, we are left with a single vertex in the splitting component. Consequently, if we now extend the partial depth-first search tree to include this vertex and obtain another depth-first search tree T' such that there is no splitting component in $G - T'$, then the largest weakly connected component in $G - T'$ has at most $\lfloor n/2 \rfloor$ vertices. Hence, using this discussion, we obtain the following theorem:

Theorem 1: Suppose a cycle separator of an n–vertex strongly connected digraph can be computed in $T_c(n)$ parallel time using $P_c(n)$ processors. Then, a depth-first search tree of G can be computed in $O(\log^2 n(T_c(n) + \log^2 n))$ time using $P_c(n) + n^3$ processors.

Proof: The above procedure uses a cycle-separator subroutine to compute a depth-first search tree, and its correctness can be seen easily. The analysis of the processor and time complexity of this procedure will be provided in the final version of the paper. ∎

4 Constructing A Directed Cycle Separator

For obtaining a directed cycle separator, we first show (cf. section 4.1) how to obtain a set of vertex disjoint directed paths $Q_f = \{q_1, ..., q_k\}$ where $k \leq 24 \log n$ such that the largest strongly connected component of $G(V - Q_f)$ has size at most $\lfloor n/2 \rfloor$, i.e., Q_f has the separating property. In section 4.2, we construct the directed path separator from Q_f and use this to obtain a cycle separator.

To obtain Q_f, suppose that at any step, we have a set of directed paths Q (where $|Q| = m$) that obey the separating property. Then, we execute a routine REDUCE (Q) that reduces the number of paths in Q by a factor of $(1 - \frac{1}{12 \log n})$, while retaining the separating property.

Initially Q consists of all vertices of V, each being a dipath of length 0; thus, the separating property holds initially. Since the size of Q is reduced by a $(1 - (12 \log n)^{-1})$ factor by each call to

REDUCE, it is readily seen that $O(\log^2 n)$ calls to REDUCE are sufficient in order to reduce the size of Q to at most $24 \log n$. Consequently, after $O(\log^2 n)$ such calls, we have the desired Q_f; we use this to obtain the desired directed cycle separator. The conversion from Q_f to a directed cycle separator is explained in section 4.2.

4.1 Reducing the Number of Paths While Maintaining the Separator Property

We now describe the main routine REDUCE. As pointed in the Introduction, the basic idea used in this subroutine is similar to that used by Aggarwal and Anderson [AA87]; they used it to obtain a depth–first search tree of an undirected graph. However, since this subroutine differs in some very crucial aspects, we describe it in detail below.

The general situation in REDUCE is having a set of vertex disjoint dipaths, Q, that obey the separating property. Q is divided into two sets of paths, L and S (where $|L| = \frac{m}{3*\log n}$ and $S = Q - L$), and the subroutine operates for $O(\log^2 m)$ phases. In any phase, only some of the paths of L are *active* and a set of vertex disjoint dipaths $P = \{p_1, ..., p_\alpha\}$that begins from the active paths of L and the *lower segments* of S. For each dipath, p_i,the starting vertex is a vertex belonging to an active path in L, the last vertex is a vertex that belongs to a *lower segment* of some dipath in S, and the interior vertices are taken from $G(V - Q)$. At the beginning of the subroutine, all paths in L are active. Each dipath in Q contains the endpoint of at most one path of P and, of course, there may be dipaths in Q that do not contain any endpoints of the paths in P. Let L_{active} (abbreviated as L_a) be the set of active paths in L, and let S_u (and S_l) denote the set of upper segments (lower segments, resp.) of the paths in S. Furthermore, let a dipath, p, join dipaths $l \in L_a$ and $s \in S_l$, and let p have endpoints x and y where $l = l'xl''$ and $s = s'ys''$. Then, during the execution of REDUCE, l is replaced by $l'ps''$, s is replaced by s', and either l'' is added to the inactive paths in L, $L_{inactive}$ (abbreviated as L_{in}), or l'' is discarded. (The conditions under which l'' is discarded and those under which l'' is added to L_{in} will be discussed later). Irrespective of whether l'' is discarded or added to L_{in}, note that the dipath, s, has been reduced to half its original length (since ys'' has at least as many vertices as s'). Furthermore, the paths in L_a and those in S did not increase in number; in fact, the number of paths in L_a and S might have decreased if some paths of P were joined to the lowest endpoints in S_l. However, the number of paths in L_{in} may increase which may, in turn, lead to an increase in the total number of paths in $L = L_a \cup L_{in}$. If the cardinality of S does not decrease then the increase in $|L|$ can increase the number of paths in $Q = L \cup S$. Consequently, below, we explain the mechanism of actually reducing the number of dipaths in Q rather than increasing it.

For a directed path p from l to s, we assign a cost equal to the length of the segment of l that is cut off, i.e., if $l = l'xl''$ where x is an end point of p, then we assign it a cost equal to the number of vertices in l''. The set, P, of vertex disjoint dipaths that subroutine REDUCE finds is the one that minimizes the total cost and has the property that there are no dipaths from L_a to S in $G(V - (L \cup S \cup P))$. In other words, REDUCE finds a minimum cost, maximal set of vertex disjoint dipaths, $P = \{p_1, ..., p_\alpha\}$ where the cost is assigned as given above. Let L_a^* be the set of all segments of the kind l'', i.e., let L_a^* be the set of all paths of L_a such that $l'' \in L_a$ if $l = l'xl'' \in L$ and l is joined by some path in P to a path in S_l. Let $L_{a,new}$ be the set of all paths of the kind $l'ps''$ and let $\hat{L}_a$ be the set of all dipaths that were in L_a but that could not be connected to any

dipath in S_l using the vertices and arcs of $G(V-Q-P)$. Similarly, let $\hat{S}_l$ be the lower segments of the dipaths in S_l that could not be connected to any path in L_a using the vertices and arcs of $G(V-Q-P)$, let $\hat{S}_u$ be the set of the corresponding upper segments of $\hat{S}_l$, and, let S_{new} be the set of all dipaths in S such that s' belongs to S_{new} if some path $s = s'ys'' \in S$ was connected to $l = l'xl'' \in L_a$.

Claim 1: If $W = V - Q - P$ then either the number of nodes in the strongly connected components of $G(W \cup L_a^* \cup \hat{L}_a)$ or that in the strongly connected component of $G(W \cup \hat{S}_l)$ is at most $\lfloor n/2 \rfloor$.
Proof: Since P is the set of minimum cost maximal set of vertex disjoint dipaths, no path in L_a^* could be joined to $\hat{S}_l$ using the vertices of $G(W)$. For if there were such a path then a set of dipaths with the same size as P could have been found that went from L_a to $\hat{S}_l$ and that had strictly less cost. Similarly, if a dipath in $\hat{L}_a$ could be joined to $\hat{S}_l$ using the vertices of $G(W)$ then P would no longer be maximal which implies that no path in $\hat{L}_a$ could be joined to $\hat{S}_l$ using the vertices of $G(W)$. Consequently, $L_a^* \cup \hat{L}_a$ and $\hat{S}_l$ fall into different strongly connected components of W when they are added to W. This implies that if the largest strongly connected component of $G(W \cup \hat{S}_l)$ has at least $\lfloor n/2 \rfloor + 1$ nodes then the largest strongly connected component of $G(W \cup L_a^* \cup \hat{L}_a)$ has at most $\lfloor n/2 \rfloor$ vertices, and vice versa. ∎

In view of Claim 1, observe that if the size of the strongly connected component of $G(W \cup \hat{S}_l)$ is at most $\lfloor n/2 \rfloor$ then we can discard $\hat{S}_l$ (i.e., add $\hat{S}_l$ to W), add $L_a^* \cup \hat{L}_a$ to L_{in} so that $L_{in,new} = L_{in} \cup L_a^* \cup \hat{L}_a$ and in the next phase work with $L_a = L_{a,new}$ and $S = S_{new} \cup \hat{S}_u$. We will call this situation case 1, and we note that all the paths in S have been reduced to half their original size: a path in S_{new} is half the size of its original path because it is a subpath of the lower segment of some path in S, and the paths in $\hat{S}_u$ are simply the upper segments whose lower segments have been discarded. Furthermore, the number of paths in L_a have not increased and those in L_{in} have increased by at most $|L_a|$.

On the other hand, if the size of the largest strongly connected component in $G(W \cup \hat{S}_l)$ is at least $\lfloor n/2 \rfloor + 1$ then using Claim 1 we discard $L_a^* \cup \hat{L}_a$ (i.e., add $L_a^* \cup \hat{L}_a$ to W) and for the execution of the next phase, we take $S = S_{new} \cup \hat{S}$ where $s \in \hat{S}$ if $s = s_l \cup s_u, s_l \in \hat{S}_l$ and $s_u \in \hat{S}_u$. That is, we take $L_a = L_{a,new} \cup L_{in,a}$ where $L_{in,a}$ is a subset of any paths from L_{in} such that $L_{in,a} = L_{in}$ when $\frac{m}{3\log n} - |L_{a,new}| \geq L_{in}$ and $|L_{in,a}| = \frac{m}{3\log n} - |L_{a,new}|$, otherwise. Now, if $|L_a^*| \geq |L_a|/3$ then we will refer to this situation as case 2; otherwise, $|L_a^*| < |L_a|/3$ and we will refer to this situation as case 3. Recall that to the beginning of REDUCE, $|L_a| = |L| = \frac{m}{3\log n}$. Consequently, after the execution of every phase if the number of paths in L_a drops below $\frac{m}{3\log n}$, then we take enough paths from L_{in} to restore L_a to its original cardinality of $\frac{m}{3\log n}$.

Claim 2: After executing $O(\log^2 m)$ phases in REDUCE, we obtain a set of dipaths Q' that obeys the separating property and for which $|Q'| \leq m(1-(12\log n)^{-1})$.
Proof: At the beginning of REDUCE (Q), $|L| = |L_a| = \frac{m}{3\log n}$ and $S = Q - L$. Let j,k, and l be respectively the number of phases in which cases 1, 2, and 3 occur. It can be readily seen that all the paths in S are halved with respect to their length in every phase of REDUCE for which case 1 occurs. Furthermore, for this case, the paths in L_{in} and, therefore, in L increase in number by at most $\frac{m}{3\log n}$. Consequently, if case 1 occurs in j phases and if $j \geq \lfloor \log n \rfloor + 1$ then after these phases the number of paths in S is at most half the original cardinality of S at the beginning of REDUCE.

That is, after $j \geq \lfloor \log m \rfloor + 1$ such phases, $|S| \leq (1/2) * \{m * (1 - \frac{1}{3 \log n})\}$. Also, since the cardinality of L at the beginning of REDUCE was $(\frac{m}{3 \log n})$, after $\lfloor \log n \rfloor + 1$ such phases, the cardinality of L is at most $(\lfloor \log n \rfloor + 1) * 1/3 * \frac{m}{\log n} \leq \frac{m}{3} + \frac{m}{3 \log n}$. Hence, after $\lfloor \log n \rfloor + 1$ phases for which case 1 occurs, the total number of paths in $Q \leq \frac{m}{2} - \frac{m}{6 \log n} + \frac{m}{3} + \frac{Q}{3 \log n} \leq \frac{5m}{6} + \frac{m}{6 \log n} \leq m - \frac{m}{12 \log n}$ when $m \geq 8$.

In cases 2 and 3, the numbers of paths in L and in S never increase. So, if there k phases for which case 2 occurs, then at least $(1/3) * (\frac{m}{3 \log n})$ paths in S are reduced to half their size for each occurrence of phase 2. Now, if $k \geq 9 \lfloor \log^2 n \rfloor + 1$ then it can readily be seen that after k such phases, the cardinality of S is at most $\frac{m}{2}(1 - \frac{1}{3 \log n})$. Also, because $j \leq \lfloor \log m \rfloor$ (otherwise claim 2 holds from the above argument), the cardinality of L is at most $\lfloor \log n \rfloor * (1/3) * \frac{m}{\log n}$. Consequently, the total number of paths is again reduced to $m * (1 - (12 \log n)^{-1})$.

Now, in every phase when case 3 occurs, the cardinality of S does not increase but that of $L = L_a \cup L_{in}$ decreases by at least $(1/3) * (m/3 \log n)$. Since $j \leq \lfloor \log m \rfloor$, the maximum cardinality of L during the execution of the algorithm is at most $\lfloor \log m \rfloor * \frac{m}{3 \lfloor \log n \rfloor}$, i.e., at most $\frac{m}{3}$. Consequently, if $m \geq \lfloor \log n \rfloor + 1$, then the maximum number of paths in $L \leq \frac{m}{3} - (\lfloor \log Q \rfloor) * (2/3) * (m/3 \log n) \leq (2/9) * \frac{m}{\log n}$. Now, at the beginning of REDUCE, L had $\frac{m}{3 * \log n}$ paths, and the number of paths S never increase during the execution of REDUCE. So, the total number of paths have been reduced by at $m/9 \log n$.

Thus after $j + k + l$ phases, i.e., after $9 \lfloor \log^2 m \rfloor + 2 \lfloor \log n \rfloor$ phases, the cardinality of m is reduced by at least $m/12 \log n$ paths. ∎

Since every call to subroutine REDUCE reduces the number of paths in Q by a factor of $1 - (12 \log n)^{-1}$, clearly, $O(\log^2 n)$ calls to REDUCE are sufficient to obtain $Q_f = \{q_1, ..., q_k\}$ where $k \leq 24 \log n$ and where Q_f obeys the separating property. Now, to obtain the desired Q_f, only need to show how to find a maximal set of disjoint paths that have the minimum cost and that go from L_a to S (where $L_a \cup L_{in} = L$ and $L \cup S = Q$). Depending on whether we use the result of Aggarwal and Anderson [AA87] or that of Goldberg, Plotkin, and Vaidya [GPV88], we obtain two different parallel algorithms; the time and processor complexities are given in Theorems 2 and 3, respectively.

Claim 3: The problem of finding a maximal set of minimum cost dipaths from L_a to S can be reduced to the problem of finding a minimum weight perfect matching in an $O(n)$–vertex digraph in which every arc has a weight at most n.

Proof: Idea: the proof of claim 3 is obtained in two parts. First we find the maximum number of vertex disjoint dipaths from L_a to S. Using Proposition 1 of [AA87], it can be shown that the problem of finding a maximum set of vertex disjoint dipaths can be reduced to that of finding the minimum weight perfect matching in some digraph G'' (with $O(n)$ nodes) in which every arc has a weight zero or one only. Next, we note that by using Proposition 2 of [AA87], the problem of finding a set of disjoint dipaths that has a given size abd the minimum cost, can be reduced to that of finding a minimum weight perfect matching in an $O(n)$–vertex graph in which every edge has a weight at most n. The only subtlety is that unlike [AA87], the graph is directed. However, this does not create any major problem and the corresponding modification will be provided in the final version of the paper. ∎

Theorem 2: Let $P_{mm}(n)$ and $T_{mm}(n)$ denote the number of processors and the parallel time

required to compute a minimum weight maximum matching of any n–node graph that has an integer weight of at most n on each of its edges. Then, subroutine REDUCE can be used to obtain a set of at most $24 \log n$ paths that obeys the separating property using $P_{mm}(n) + n^3$ processors and $O(\log^4 n * (T_{mm}(n) + \log^2 n))$ time.

Proof: Recall that $O(\log^2 n)$ calls to REDUCE are sufficient in order to obtain the required set of separating dipaths. Furthermore, REDUCE has $O(\log^2 n)$ phases and each phase requires two major computations — (1)computing the strong and weak components of a digraph with at most n vertices and (2)computing a maximal set of disjoint dipaths from L_a to S such that the set has minimum cost. Since the strong and weak components of an n–node digraph can be computed in $O(\log^2 n)$ time using n^3 processors, Theorem 1 follows after incorporating claim 3. ∎

Theorem 3: The problem of finding a maximal set of vertex disjoint paths (that have the minimum cost) from L_a to S in a directed graph, can be solved in $O(\sqrt{n} * \log^4 n)$ time using n^3 processors.

Proof: Follows easily from the corresponding result of [GPV88] for undirected graphs. ∎

4.2 Constructing a Cycle Separator from a Small Set of Separating Dipaths

Given a set of vertex disjoint dipaths $Q_f = \{q_1, ..., q_k\}$ where $k \leq 24 \log n$, we show below how to construct a dipath separator. Later in this subsection, we will use this dipath separator to construct a cycle separator. The subroutine for constructing a dipath separator is essentially a sequential algorithm; the only use of parallelism is in low level routines such as determining strongly and weakly connected components of a digraph and in finding a directed path that connects one dipath to another. The subroutine consists of $k - 1$ stages. During the entire execution of the subroutine, we keep a dipath, q; in the beginning $q = q_1$, after the execution of the subroutine q is the required separating dipath and, in the i–th stage q is "combined" with q_{i+1} such that the resulting q along with $\{q_{i+2}, ..., q_k\}$ obeys the separating property. To complete our description of the subroutine, we explain below the "combination process" between q and q_{i+1} that occurs in the i–th stage; this combination process is essentially the same as that provided in section 4.1.

Let $q_{i+1,l}$ and $q_{i+1,u}$ denote the lower and the upper segments of q_{i+1}, respectively. If the size of the largest strongly connected component in $G(V - \{q, q_{i+1,l}, q_{i+2}, ..., q_f\})$ is at most $\lfloor n/2 \rfloor$ then we discard $q_{i+1,l}$ (i.e., add $q_{i+1,l}$ to $V - \{q, q_{i+1}, q_{i+2}, ..., q_k\}$) and refer to this situation as case (a). In this case, q will be unchanged but $q_{i+1,u}$ will become the new q_{i+1}. Otherwise, we find the topmost vertex of q that has a dipath to $q_{i+1,l}$ using the vertices and arcs of $G' = G(V - \{q, q_{i+1}, q_{i+2}, ..., q_k\})$. If there is no dipath from any vertex of q to a vertex of $q_{i+1,l}$, then using ideas similar to Claim 1, it can be seen that the largest strongly connected component in $G(V - \{q_{i+1}, q_{i+2}, ...q_k\})$ has at most $\lfloor n/2 \rfloor$ vertices. Consequently, in this case, we discard q (i.e., add it to $(V - \{q, q_{i+1}, ..., q_k\})$), and call this situation case (b). Otherwise, let x be the topmost vertex of q that has a dipath to $q_{i+1,l}$, and let q^* be the dipath of the vertices of q above x. Then, again using ideas similar to Claim 1, it can be seen that the largest strongly connected component in $G(V \cup q^* - \{q, q_{i+1}, ..., q_k\})$ is at most $\lfloor n/2 \rfloor$. Consequently, we discard q^*, and if p denotes dipath that connects q to $q_{i+1,l}$ (and if p has endpoints x and y) then we let the new q be $q'pq''_{i+1}$ (where q' and q''_{i+1} are the segments of the paths of q and q_{i+1} that are below x and above y respectively), and we let the new q_{i+1} equal $q_{i+1} - q''_{i+1}$. We refer to this situation as case (c).

Now, if case (b) occurs then q is eliminated and q_{i+1} becomes the new q. On the other hand, q_{i+1} is reduced to at least half its previous size if either case (a) or case (c) occurs. Consequently, if we execute the process $\lfloor \log |q_{i+1}| \rfloor + 1$ times, i.e., $O(\log n)$ times, then q_{i+1} ceases to exist in Q_f. Thus, $k-1$ stages are sufficient (where each stage consists of repeating the above process at most $O(\log n)$ times) to yield a directed separator, which, in turn, yields the required algorithm for finding a directed separator in a given directed graph.

Theorem 4: Let $P_{mm}(n)$ and $T_{mm}(n)$ denote the processor and time complexity for computing a minimum weight maximum matching of any n–vertex digraph. Then, a cycle separator of an n–digraph $G = (V, A)$ can be found in $O(\log^4 n(T_{mm}(n)+\log^2 n))$ time and using $P_c(n)+n^3$ processors.
Proof: From Theorem 2, we know that $O(\log^4 n(T_{mm}(n)+\log^2 n))$ time and $P_c(n)+n^3$ processors are sufficient for obtaining a set $Q_f = \{q_1, ..., q_k\}$ of dipaths where $k \leq 24 \log n$ and Q_f obeys the separating property. Also, using the above procedure, we can obtain a dipath separator from Q_f in $O(\log^2 n)$ phases where, in each phase we find a dipath from the highest possible vertex of one dipath to another dipath in a suitable digraph. Now, since we can find such a dipath by using transitive closure, we can obtain a directed path separator from Q_f in $O(\log^2 n)$ time using n^3 processors. Consequently, a directed path separator can be obtained in a total of $O((T_{mm}(n) + \log^2 n) \log^4 n)$ time by using $P_c(n) + n^3$ processes.

Now, let $P = \{v_1, ..., v_s\}$ denote the directed path separator in the digraph G. Then, it is easy to obtain v_r such that $\{v_1, ..., v_r\}$ forms a directed path separator but $\{v_1, ..., v_{r+1}\}$ does not, by computing the transitive closure of a suitable $O(n)$–vertex digraph. Finally, we can also find the smallest index t such that v_r can reach v_t in $O(\log^2 n)$ time by using n^3 processors; this yields the desired processor and time bounds. ∎

5 Discussion

Using Theorems 1, 2, 3, and 4, we can now state the main results of this paper:

Theorem 5: Let $T_{mm}(n)$ and $P_{mm}(n)$ denote the time and the number of processors to compute a minimum weight maximum matching of any n–vertex directed graph where arcs have weights that are integers between 1 and n. Then, a depth-first search tree of an n–vertex digraph, $G = (V, A)$, can be computed in $O(\log^6 n(T_{mm}(n) + \log^2 n))$ time by using $(P_{mm} + n^3)$ processors.
Proof: Follows from Theorems 1 and 4. ∎

Theorems through 5 and their proofs have several implications, two of which are listed below:

Corollary: (a) If $M(n)$ denotes the sequential time to multiply two $n * n$ matrices then there is a randomized parallel algorithm that computes a depth-first search tree of an n–vertex graph in $O(\log^8 n)$ time and uses $nM(n) + n^3$ processors. (b) A depth-first search tree of an n–vertex directed graph can be found deterministically in $O(\log^{12} n * \sqrt{n})$ time by using n^3 processors.
Proof: Mulmuley, Vazirani, and Vazirani [MVV87] have provided a randomized parallel algorithm such that $T_{mm}(n) = O(\log^2 n)$ and $P_{mm}(n) = nM(n)$. This yields Corollary (a). Also, in Theorem 4, in order to compute a path separator or a cycle separator, we only needed to compute a

minimum weight maximal set of paths where arcs have weights that are integers between 0 and n. Consequently, we can use Theorem 3 to obtain Corollary (b). ∎

Finally, this paper leaves the following problems unresolved:

(1) Can a depth-first search tree of an undirected graph be computed in NC?

(2) Is there a deterministic algorithm for finding the minimum weight maximal set of vertex disjoint paths (as defined in section 3) in $o(\sqrt{n})$ parallel time? Such an algorithm would improve Corollary (b) and may also help in providing insight into the depth-first search problem.

(3) Does there exist a random or deterministic algorithm that has optimal or near optimal processor-time complexity and that computes a depth-first search tree in sublinear time?

6 References

[AA87] A. Aggarwal and R. Anderson, "A Random NC Algorithm for Depth-First Search," Proc. 19^{th} ACM Symposium on Theory of Computing, pp. 325-334, 1987.

[An85] R. Anderson, "A Parallel Algorithm for the Maximal Path Problem," Proc. 17^{th} ACM Symposium on Theory of Computing, pp. 37-47, 1985.

[GB84] R. K. Ghosh and G. P. Bhattacharjee, "A Parallel Search Algorithm for Directed Cyclic Graphs," BIT, Vol. 24, pp. 134-150, 1984.

[Sm86] J. R. Smith, "Parallel Algorithms for Depth-First Search, I. Planar Graphs," SIAM J. of Computing, Vol. 15, No. 3, pp. 814-830, Aug. 1986.

[EA77] D. Eckstein and D. Alton, "Parallel Graph Processing Using Depth First Search," Proc. of the Conference on Theoretical Computer Science at the University of Waterloo, pp. 21-29, 1977.

[HY87] X. He, and Y. Yesha, "A Nearly Optimal Parallel Algorithm for Constructing Depth First Spanning Trees in Planar Graphs," 1987?

[MVV87] K. Mulmuley, U. V. Vazirani, and U. V. Vazirani, "Matching as Easy as Matrix Diversion," Combinatorica, Vol. ??, pp. ??, 1987.

[RC78] E. Reglibati and D. Corneil, "Parallel Algorithms in Graph Theory,' SIAM J. of Computing, Vol. 7, pp. 230-237, 1978.

[Re83] J. Reif, Depth-First Search is Inherently Sequential," Aiken Computation Lab., Tech. Report, TR-27-83, Nov. 1983.

[GPV88] A. Goldberg, S. Plotkin, and P. Vaidya, "Sublinear-Time Parallel Algorithms for Matching and Related Problems," Manuscript, 1988.

[Ka88] M. Y. Kao, "Planar Directed Depth-First Search is in DNC," To be presented at Aegean Workshop on Computing, Greece, 1988.